REMARKS

Applicant is in receipt of the Office Action mailed November 12, 2008. Claims 1, 18, 20, 28, 29, and 30 have been amended. Claims 1-25 and 28-30 are pending in the case. Reconsideration of the present case is earnestly requested in light of the following remarks.

Allowed Subject Matter

Applicant appreciates the allowed subject matter of claims 9-13 and 18-20, but believes that the claims as currently amended are patentably distinct and non-obvious over the cited art, as explained below.

Section 112 Rejections

Claims 1-25 and 28-29 were rejected under 35 U.S.C. 112, second paragraph, as being indefinite. Applicant has amended independent claims 1, 28, 29, and 30, as well as dependent claims 18 and 20, accordingly, numerically labeling the higher level claim clauses to clearly distinguish from the alphabetically labeled nested claim clauses. Applicant respectfully notes that these labels are intended for identification purposes only, and are not intended to imply any particular ordering of the claim elements beyond that implied by the claim language. Applicant has also amended these claims to clarify that performing the program exit procedure terminates execution of the program (or subprogram). Finally, Applicant has further amended means plus function claim 29 to make the claim more readable. Applicant thus respectfully requests removal of the section 112 rejection of the claims.

Section 103 Rejections

Claims 1-8, 14-20, and 22-30 were rejected under 35 U.S.C. 103(a) as being unpatentable over "LabVIEW Real-Time Module User Manual" ("LabVIEW RT"), in view of Fuchs et al. (US Patent No. 5,530,802, "Fuchs"). Applicant respectfully traverses the rejection.

Amended claim 1 recites:

1. A computer-accessible memory medium that stores program instructions for performing time-bounded execution of a program, wherein the program instructions are executable by a processor to perform:

1) initiating a timed program execution process, wherein the timed program execution process is operable to execute a program in a time-bounded manner;

2) initiating a timeout process, wherein the timeout process is operable to preempt the execution process to interrupt execution of the program;

3) configuring a timeout event, wherein the timeout event is an event indicating a timeout condition for the program;

4) the timed program execution process performing a time-bounded execution of the program, comprising:

a) determining and storing a rollback state for the program;

b) if the timeout event has not occurred, executing the program, wherein, during said executing, if the timeout event occurs,

c) the timeout process setting the timed program execution process to the rollback state, and disabling the timeout event; and

d) the timed program execution process resuming executing the program based on the rollback state with a timeout condition in preparation to perform a program exit procedure; and

e) performing the program exit procedure, thereby terminating execution of the program;

5) disabling the timeout event;

6) terminating the timeout process; and

7) terminating the timed program execution process.

As explained in the previous Response, which is hereby incorporated by reference, in Applicant's claimed invention as represented in claim 1, a program is required to execute within a determined duration, but in case of a program time-out, it is desired that the program exit, but not be left in an unstable state. Thus, a roll-back state is determined for the program, where it is known that in the roll-back state the program is stable. Then, during execution of the program, if the program times out, the program is reset to the roll-back state (thus, putting the program into a stable state), and allowed to exit gracefully with a time-out condition (thereby terminating execution of the program). Thus, while the program is terminated, this termination occurs such that the program exits gracefully, i.e., in a stable state.

Applicant respectfully notes that LabVIEW RT is directed to developing real time graphical programs, e.g., deterministic applications, for implementation or execution on specialized hardware, e.g., RT Series hardware provided by National Instruments Corporation, where the real time performance is implemented via executing tasks in prioritized threads, and is not germane to the particular techniques recited in claim 1.

Fuchs is directed to automatically working around a failure due to a fault detected during program execution. In response to detecting the fault, the method of Fuchs resets to a rollback point, then iteratively attempts to bypass the fault by reordering inputs to the program, where if the first reordering doesn't avoid the fault, a second reordering is attempted, and so on.

In direct contrast to LabVIEW RT and Fuchs, per claim 1, the timed program execution process controls the program execution, where the timed program execution process and the timeout process are separate and distinct, e.g., they may execute in different threads, and where the timeout process affects the program execution process (which isn't taught by LabVIEW RT)–not simply switching threads (stopping the process from running), but rather, it changes the program execution process, resetting it to a previous state, specifically, the rollback state, and exiting gracefully with a timeout condition. There are numerous benefits to this approach (which are not provided by the cited art), e.g., in addition to ensuring a graceful exit from the execution process, the rollback state may be specified to ensure a valid state for system resources, e.g., allocated memory, processor mode/state. Fuchs, which is directed to software failure recovery, not

time-bounded program execution, also fails to teach or suggest this functionality, as does the combination of LabVIEW RT and Fuchs, as will now be explained.

Applicant recognizes that there are similarities between some terms in the cited references and Applicant's claim language, but does not believe that the combination of the cited references results in the specific functionality recited in Applicant's independent claims, summarized above. For example, Applicant notes that in Fuchs's fault bypass system and reorder recovery algorithm, the recovery process is performed in response to an application crashing or hanging, and results in restoring the application process to a checkpoint, reordering inputs logged since the checkpoint, and reprocessing the reordered inputs, i.e., continuing the application process by bypassing the fault, which Applicant believes is quite different from Applicant's graceful *program recovery and exit*. Nor does combining Fuchs's fault bypass system with LabVIEW RT's approach using prioritized execution threads/VIs produce Applicant's claimed functionality, at least since, as noted above, LabVIEW RT's system manages task execution via thread priorities, but does not operate in the particular manner recited in claim 1 when a task fails to complete within a specified time, as explained in more detail below. Nowhere does Fuchs (or LabVIEW RT) disclose rolling back to a rollback point, then exiting gracefully with a timeout condition, all in response to a *failure to execute a program within a specified time*.

Considering the claimed features more specifically, Applicant submits that there are numerous limitations of claim 1 that are not taught or suggested by the cited art.

For example, nowhere does the cited art disclose **4) the timed program execution process performing a time-bounded execution of the program, comprising: a) determining and storing a rollback state for the program; b) if the timeout event has not occurred, executing the program, wherein, during said executing, if the timeout event occurs, c) the timeout process setting the timed program execution process to the rollback state, and disabling the timeout event; and d) the timed program execution process resuming executing the program based on the rollback state with a timeout condition in preparation to perform a program**

exit procedure; and e) performing the program exit procedure, thereby terminating execution of the program, as recited in claim 1.

Cited p.4-4 of LabVIEW RT is directed to dividing tasks to create deterministic multithreaded applications, as acknowledged by the Examiner, where time-critical tasks are separated from non-time-critical tasks and put in a separate VI so that adequate processing resources may be allocated to them. Note, however, that per lines 15-19, the time-critical VI receives the processor resources until either it yields to another VI or it completes the task. Note that in LabVIEW RT, if the high priority VI yields to another VI, eventually the processor resources will again be returned to that priority VI where upon it will resume where it left off, so that the high priority VI is guaranteed to complete its high priority tasks. This is emphasized by the text quoted in the Office Action, "The process repeats until all tasks complete."

In direct contrast, in claim 1, the program is not allowed to complete its specified functionality, but rather is interrupted by the timeout event, reset to an earlier state, specifically the determined rollback state for the program, from which it resumes with a timeout condition, which results in the program exiting, i.e., terminating. Thus LabVIEW RT actually teaches away from claim 1.

In asserting that LabVIEW RT teaches method element d) of claim 1, the Office Action cites p.5-16, lines 1-8, which refers to using the API of the Real-Time FIFO VIs to detect a failure in the communication path or in one of the participants respect to network communications (Network Watchdog), where, in response to such failures, the system is shut down entirely until communication is reestablished, after which the system, including the real-time application, is returned to a known state, and continues *as usual*. Note that no "rollback" state is determined or set–the known state isif yes nowhere described as a rollback state, and furthermore, continuing *as usual* teaches away from resuming from a rollback state with a timeout condition, which results in (graceful) termination of the program execution.

This same citation is given with regards to method element e) performing the program exit procedure, thereby terminating execution of the program. Clearly, continuing *as usual* in no way teaches or suggests resuming from a rollback state with a timeout condition, which results in termination of the program execution.

The Office Action admits that LabVIEW fails to disclose a) determining and storing a rollback state for the program, and (in response to a timeout event during execution) c) the timeout process setting the timed program execution process to the rollback state, and disabling the timeout event, but asserts that Fuchs remedies this admitted deficiency of LabVIEW, citing col.2:51-56, and col.3:24-30. Applicant respectfully disagrees.

As explained above and in the previous Response, Fuchs's approach is fundamentally different from Applicant's claimed technique, and is not designed for the same purpose. More specifically, repeating the above, in response to detecting the fault, Fuchs resets to a rollback point, then iteratively attempts to bypass the fault by reordering inputs to the program, where if the first reordering doesn't avoid the fault, a second reordering is attempted, and so on.

Cited col.2:51-56 reads:

> The logging of inputs received by each application process in the input log effectively places a "logical" checkpoint at the end of the ensuing state interval, which serves to minimize the domino effect of rollback propagation which would tend to include other processes in the recovery in a multiple process environment.

Cited col.3:24-30 reads:

> In one embodiment, a reorder recovery algorithm is provided that consists only of the receiver reorder step of the overall progressive retry algorithm. When information is known about the particular application process or the cause of the detected fault, it may be determined that the reorder recovery algorithm may be more appropriate for bypassing the fault.

As may be seen, this text in no way teaches or suggests "c) the timeout process setting the timed program execution process to the rollback state, and disabling the timeout event", contrary to the Office Action's assertion. Rather this citation simply discloses provision of a reorder recovery algorithm consisting only of a receiver reorder step in an overall progressive retry algorithm, whose use may be appropriate for bypassing faults in an application process. As Fuchs states in the Abstract, the reorder recovery algorithm attempts "to bypass the detected fault by reordering and reprocessing the inputs that have been received by the faulty application process", and has nothing to

do with timed program execution. More specifically, this citation makes no mention of a timeout process setting a timed program execution process to a rollback state (in response to a timeout event), nor disabling the timeout event.

As mentioned in the previous Response, Fuchs nowhere even mentions a timeout process at all, nor a timeout event, and more specifically fails to disclose a timeout process rolling back a timed program execution process to a rollback state in response to a timeout event, and then disabling the timeout event.

Thus, the cited art fails to teach or suggest these features of claim 1.

Nor does the cited art disclose **terminating the timeout process**, as recited in claim 1.

In asserting that LabVIEW RT discloses this feature, the Office Action quotes p.5-16:4-8, specifically, "For example, you can check if the RT FIFO is unexpectedly empty.....returning to a known state, and continuing as usual.", which clearly does not teach terminating a timout process. Nowhere does this citation, nor LabVIEW RT in general, teach terminating a timout process after termination of program execution, as claimed. Rather, in LabVIEW RT, the VI is *always* allowed to complete its tasks.

Thus, the cited art fails to teach or suggest this feature of claim 1.

Nor does the cited art disclose **terminating the timed program execution process**, as recited in claim 1.

In asserting that LabVIEW RT discloses this feature, the Office Action quotes p.5-14, specifically, "In control applications, it might be necessary to respond to a failure or.....failure must be handled in the same manner.", which clearly does not teach terminating a timed program execution process. In fact, this text simply states that different actions may be taken depending on the type of failure, including shutting down equipment, and continuing to operate until network communications are reestablished, neither of which teaches terminating a timed program execution process.

Thus, the cited art fails to teach or suggest this feature of claim 1.

As discussed above at length, the timed program execution process resuming executing the program based on the rollback state with a timeout condition allows the program to end gracefully, e.g., without leaving the system or its resources in unstable or inappropriate states, a functionality not provided or described by LabVIEW RT and/or Fuchs.

Thus, the cited art fails to teach or suggest these claimed features.

Thus, for at least the reasons provided above, Applicant submits that LabVIEW RT and Fuchs, taken singly or in combination, fail to teach or suggest all the features and limitations of claim 1.

Applicant further submits that the Office Action has not provided a proper reason to combine LabVIEW RT and Fuchs. For example, the only motivation to combine suggested by the Office Action is "because one of the [*sic*]ordinary skills of the art would want to be able to utilize the CPU performance by using the rollback state or any application that has failed or timeout for any event".

Applicant respectfully submits that wanting "to be able to utilize the CPU performance" has no bearing on the novel and beneficial functionality of Applicant's invention as represented by claim 1, at least for the reason that "utilizing the CPU performance" is not germane to resuming executing the program based on the rollback state with a timeout condition (thereby exiting the program execution gracefully). In other words, a primary benefit of the functionality recited in claim 1 is that when a timeout occurs, and the program is rolled back and exits with a timeout condition, the system is prevented from ending up in an undesirable or unpredictable state, which is not germane to improving the performance of a multi-threaded CPU. Thus, Applicant submits that LabVIEW RT and Fuchs are not properly combinable to make a prima facie case of obviousness.

Moreover, even were LabVIEW RT and Fuchs properly combinable, which Applicant argues they are not, the resulting combination would still not produce Applicant's invention as recited in claim 1, as discussed above at length.

Thus, for at least the reasons provided above, Applicant submits that claim 1, and those claims respectively dependent therefrom, are patentably distinct and non-obvious over the cited art, and are thus allowable.

Claims 28, 29, and 30 include similar novel limitations as claim 1, and so the above arguments apply with equal force to these claims. Thus, for at least the reasons provided above, Applicant submits that claims 28, 29, and 30, and those claims respectively dependent therefrom, are patentably distinct and non-obvious over the cited art, and are thus allowable.

Claim 21 was rejected under 35 U.S.C. 103(a) as being unpatentable over LabVIEW RT in view of Fuchs, and further in view of Chamberlain (US Patent No. 6,438,749).

Applicant respectfully submits that since claim 21 depends from independent claim 1, which was shown above to be patentably distinct and non-obvious, and thus allowable, claim 21 is similarly patentably distinct and non-obvious, and thus allowable, for at least the reasons provided above.

Applicant also asserts that numerous ones of the dependent claims recite further distinctions over the cited art. However, since the independent claims have been shown to be patentably distinct, a further discussion of the dependent claims is not necessary at this time.

Removal of the section 103 rejection of claims 1-8, 14-21, and 22-30 is earnestly requested.

CONCLUSION

Applicant submits the application is in condition for allowance, and an early notice to that effect is requested.

If any extensions of time (under 37 C.F.R. § 1.136) are necessary to prevent the above-referenced application(s) from becoming abandoned, Applicant(s) hereby petition for such extensions. The Commissioner is hereby authorized to charge any fees which may be required or credit any overpayment to Meyertons, Hood, Kivlin, Kowert & Goetzel P.C., Deposit Account No. 50-1505/5150-81401/JCH.

Also filed herewith are the following items:

☐ Request for Continued Examination

☐ Terminal Disclaimer

☐ Power of Attorney By Assignee and Revocation of Previous Powers

☐ Notice of Change of Address

☐ Other:

Respectfully submitted,

/Jeffrey C. Hood

Jeffrey C. Hood, Reg. #35198
ATTORNEY FOR APPLICANT(S)

Meyertons, Hood, Kivlin, Kowert & Goetzel PC
P.O. Box 398
Austin, TX 78767-0398
Phone: (512) 853-8800
Date: 2009-01-20 JCH/MSW